

AVP-2933

Best Practices in WPS V7 Performance Tuning

Best Practices in WPS V7 Performance Tuning

What this lab is about.....	2
Lab requirements	2
What you should be able to do	2
Exercise 1 - Introduction to WebSphere Process Server tuning example.....	3
Exercise 2 – Configure WebSphere Application Server artifacts to tune WPS performance.....	16
Exercise 3 – WebSphere Process Server Memory tuning.....	34

What this lab is about

Based on an example IBM WebSphere Process Server V7.0 application you will observe the differences in performance between various Component Binding types (SCA, WS, HTTP and Messaging) and flavors (synchronous, asynchronous) available in WPS. You will configure the WAS/WPS infrastructure by manipulating performance relevant parameters (create and use new thread pools, configure module-level activation specifications, tune Java Messaging Service (JMS) and JVM Heap) and observe their impact on WPS application performance

This lab is provided **AS-IS**, with no formal IBM support.

Lab requirements

List of system and software required for the attendee to complete the lab.

- IBM WebSphere Process Server V7
- WebSphere Integration Developer
- PMAT JVM tool

What you should be able to do

At the end of this lab you should be able to

- Understand the main differences between various WPS binding types in terms of Performance
- Understand and configure the main artifacts in WebSphere Application Server that influence WPS performance
- Analyze, read and understand verbose Garbage Collection logs

Exercise 1 - Introduction to WebSphere Process Server tuning example

What this exercise is about

This exercise gives you an introduction to the WPS tuning example. This lab contains very simple WebSphere Process Server modules which you are going to use to simulate the performance aspects of different Binding types. In this exercise you'll get used to the example application and other supporting tools you will need during the lab.

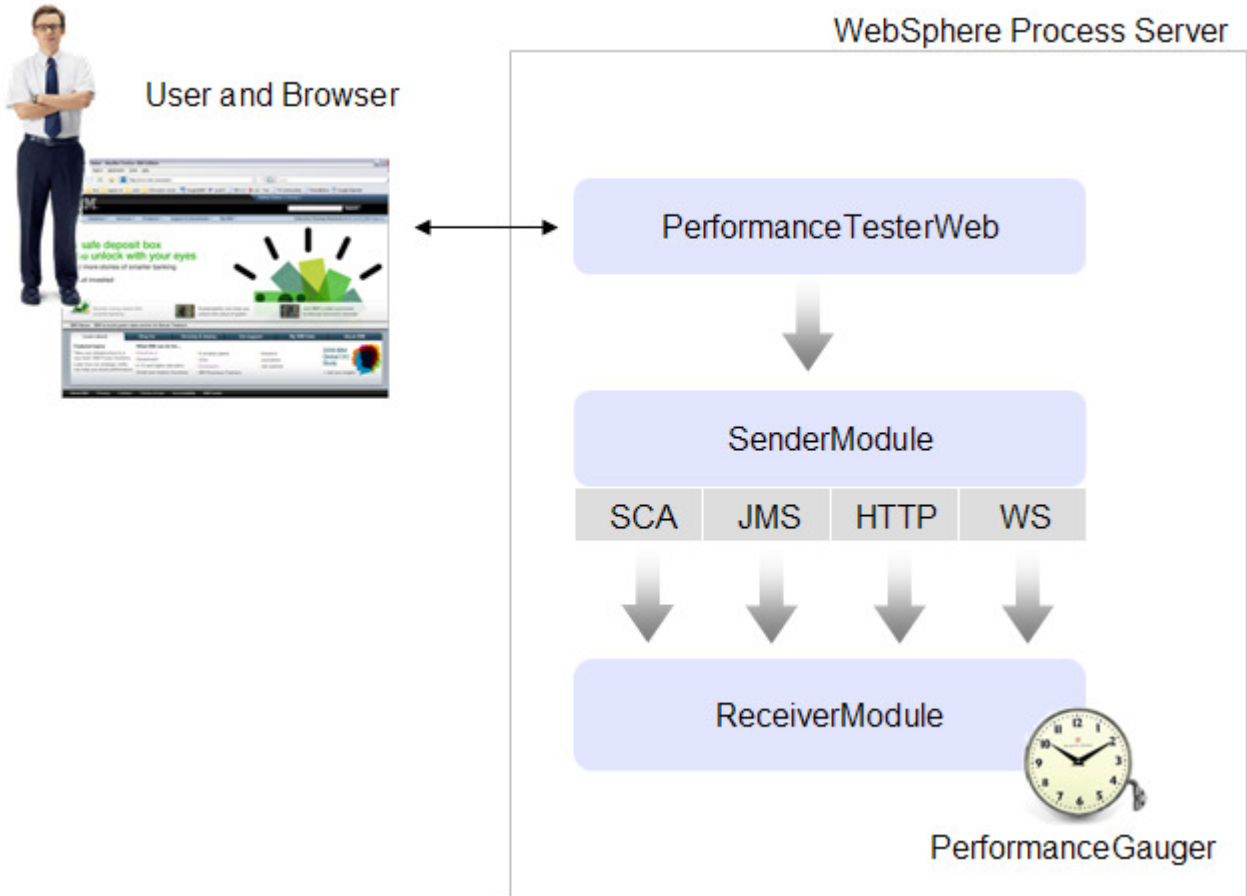
What you should be able to do

At the end of the exercise, you should be able to:

- Configure the JVM Heap and Verbose Garbage Collection
- Understand the architecture and functionality of the Performance Tester example application
- Use Wintail to monitor the WPS SystemOut log in Real-time
- Run the first tests and basic use-cases with the Performance Tester application
- Explain the impact of asynchronous communication on performance

Introduction

This lab contains WebSphere Process Server modules that will be used to configure the concurrency of different WebSphere Process Server binding styles. The actual function of the example modules is very simple and is structured as follows:



The user controls the Performance Test via the Browser which directs to the **PerformanceTesterWeb** application running on WPS. This application contains a Servlet which triggers the **SenderModule** to call the **ReceiverModule**.

The **ReceiverModule** module includes four WebSphere Process Server exports to support the different WebSphere Process Server binding styles that will be observed. The export types are SCA, Messaging, HTTP and Web Services.

**Note**

The performance results from this lab should not be considered representative of the performance of WebSphere Process Server and should not be generally published as a general communication of WPS performance capabilities.

The intention of this lab is to enable students to learn about some of the key aspects of WebSphere Process Server performance without requiring performance benchmark class systems (like Tivoli performance monitoring or others). In other words, students will be able to run this lab on their personal Laptops in their personal WebSphere Integration Developer, WebSphere Process Server and Browser environments. If you run this lab in this type of environment, you will not collect representative, publishable performance numbers. Here are some of the issues why the performance number might not be representative:

- Single processor host system
- Standard performance disk subsystem
- Limited physical memory (likely that physical memory is overcommitted, which means your machine may page)
- Possibly in a Windows VmWare environment where system resources are being shared with regular client applications
- In a development environment instead of a stand-alone WPS deployment (an improved environment would be where WID is running on a different system from WPS)
- Load generator running on a system under test
- Endpoint emulations running on a system under test

Exercise instructions

Part 1: Set up the environment

___ 1. Start the **VMware** image if it is not already running. If it is not running, follow these steps at 1.

___ a. Request assistance from the AVP staff to start the VMware image

___ b. Log in using the following user ID and password:

User ID: **Administrator**

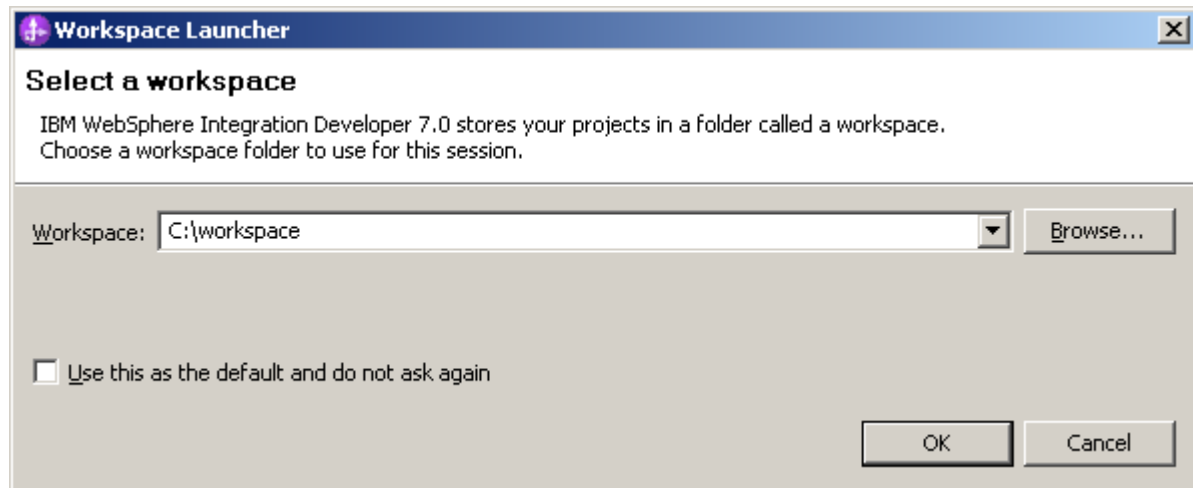
Password: **Happy2Be**

___ 2. Start WebSphere Integration Developer by double-clicking the WID icon on the Desktop

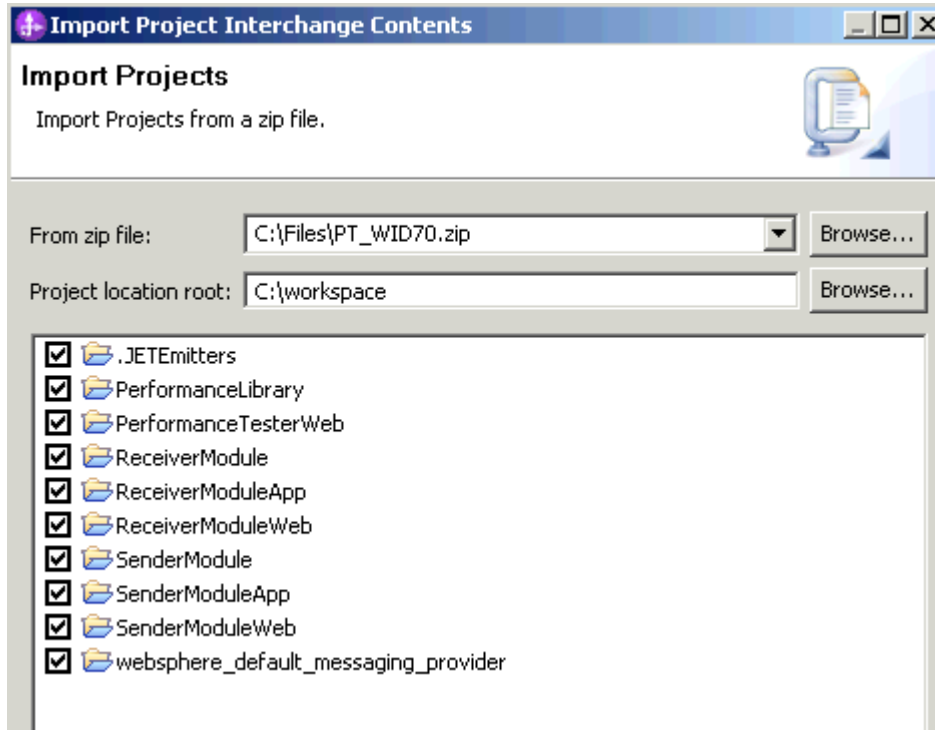


WID V7.0.0.3

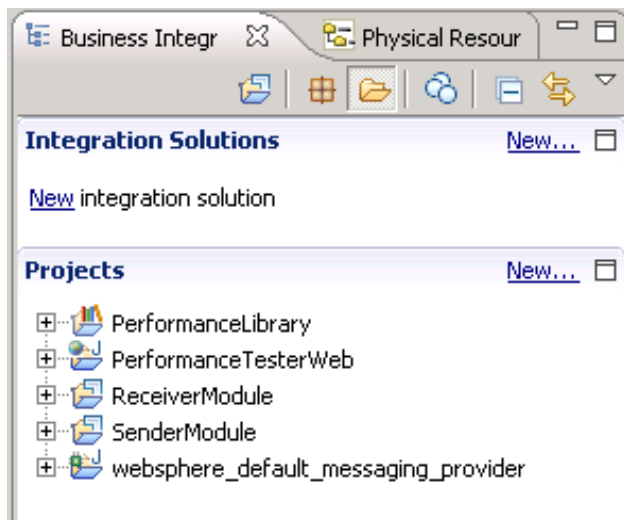
___ 3. Select the workspace to be **C:\workspace** and click **OK**.



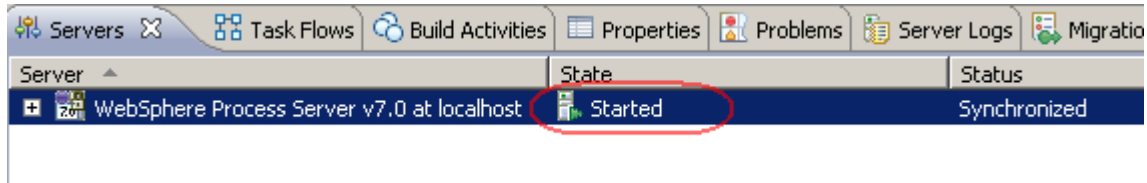
- ___ 4. Import project file by selecting **File > Import > Other > Project Interchange** and choose **C:\Files\PT_WID70.zip** > Select all Projects for the import



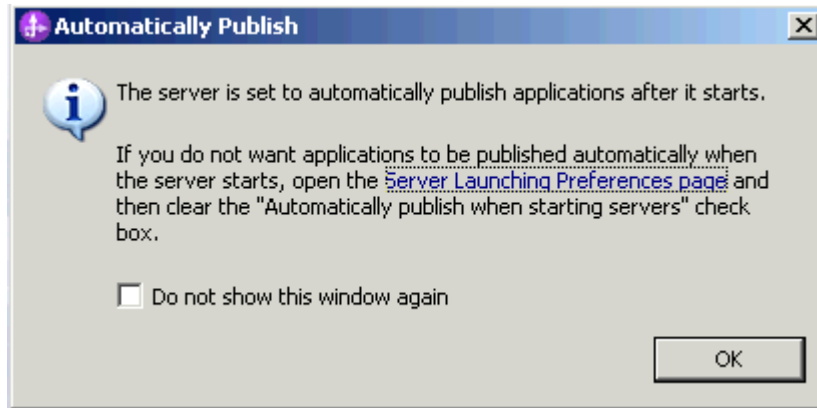
- ___ 5. The **Business Integration** view should look as shown in the following screen capture:



- ___ 6. If the server is not running already, start the server.
 - ___ a. In the **Servers** tab, right-click **WebSphere Process Server v7.0** and select **Start** from the context menu.
 - ___ b. Wait until the server starts successfully.



- ___ c. The following prompt comes up when the server is started. Select **Do not show this window again** and press OK



- ___ 7. Launch the administrative console.
- ___ a. In the **Servers** view, right-click **WebSphere Process Server V7.0** and select **Administration > Run administrative console.**

i **Information**

Alternatively, start Firefox, and select the **WPS Admin Console** bookmark or this URL:
<http://localhost:9060/ibm/console>

- ___ b. At the **Welcome** page, simply click the **Log In** button without giving any username (security is disabled in this example)
- ___ 8. Navigate through the administrative console to the Java virtual machine configuration page.
- ___ a. From the left, select **Servers > Server Types > WebSphere application servers.**
- ___ b. Select **server1.**
- ___ c. Under **Server Infrastructure**, expand the **Java and Process Management** node.
- ___ d. Select **Process Definition.**

Server Infrastructure

- Java and Process Management
 - [Class loader](#)
 - [Process Definition](#)
 - [Process Execution](#)

__ e. Select **Java Virtual Machine**.

[Application servers](#) > [server1](#) > **Process Definition**

Use this page to configure a process definition. A process definition defines the command line information necessary to start or initialize a process.

Configuration

<u>General Properties</u>	<u>Additional Properties</u>
Executable name <input type="text"/>	<input checked="" type="checkbox"/> Java Virtual Machine
Executable arguments <input type="text"/>	<input type="checkbox"/> Environment Entries
	<input type="checkbox"/> Process Execution
	<input type="checkbox"/> Process Logs
	<input type="checkbox"/> Logging and Tracing

__ f. Verify that the **Verbose garbage collection** check box is selected. If not, check the box and click **Apply**.

__ g. Set the **Maximum Heap Size** to 384 MB.

__ h. Click **OK**.

[Application servers](#) > [server1](#) > [Process Definition](#) > [Java Virtual Machine](#)

Use this page to configure advanced Java(TM) virtual machine settings.

Configuration **Runtime**

General Properties **Additional Properties**

Classpath

Boot Classpath

Verbose class loading


Verbose garbage collection

Verbose JNI

Initial Heap Size

Maximum Heap Size

[Custom Properties](#)

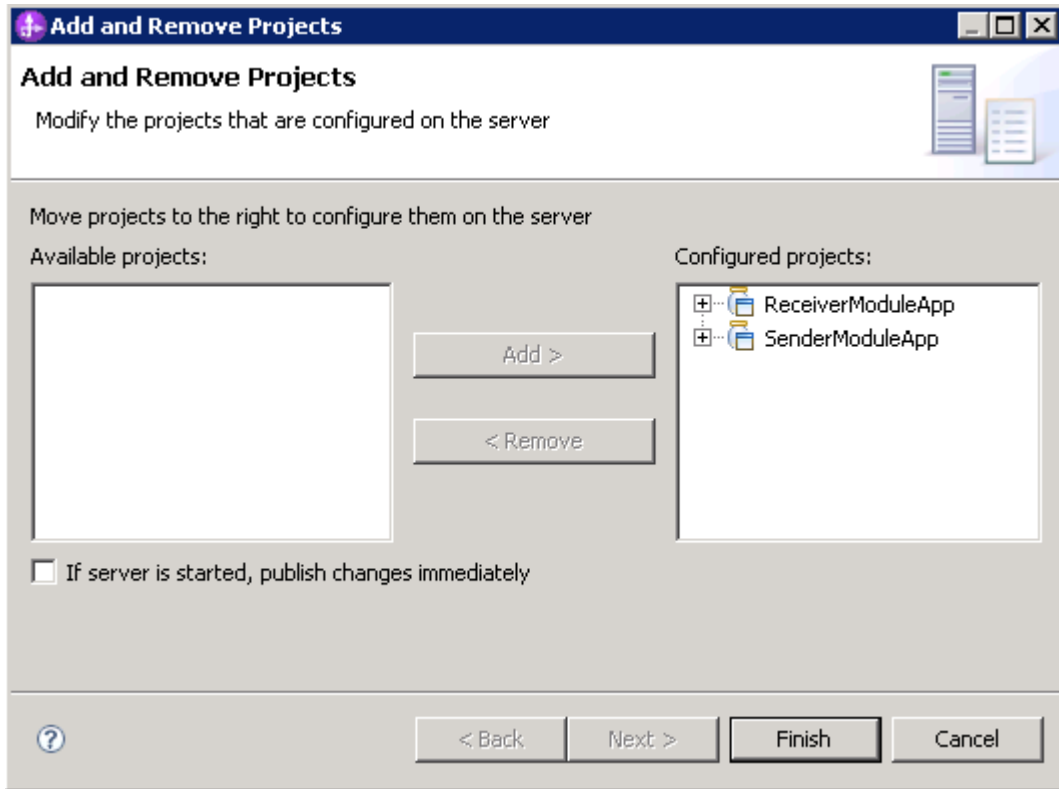
 **Note**

The default for **Maximum Heap Size** in the console is “not specified” (blank), which means the JVM default for **Maximum Heap Size** will be used. The default **Maximum Heap Size** is 512 MB. For a production environment, enter an appropriate heap size in the **Maximum Heap Size** field and select **Apply**.

- ___ i. Select the **Save** link to save.
- ___ j. Log out of the administrative console.
- ___ 9. Restart the server.
- ___ a. In the **Servers** view, right-click **WebSphere Process Server V7.0** and select **Restart**.
- ___ b. Wait until the server automatically stops and then restarts itself.

Part 2: Get used to the PerformanceTester application

- ___ 1. In WebSphere Integration Developer, add the **ReceiverModuleApp** to the server, then add the **SenderModuleApp** to the server. The Web Application Part **PerformanceTesterWeb** is deployed together with the SenderModuleApp
 - ___ a. Right-click the Server View and select 'Add and Remove Projects...'
 - ___ b. from the 'Available projects' section, add the **ReceiverModuleApp** and the **SenderModuleApp** both together to the 'Configured projects' and press Finish.



- ___ c. Both applications are now being deployed to the WPS. Observe the status in the Server view. The deployment is finished when the status changes to 'Synchronized'
- ___ 2. Open the Firefox Browser by using the Windows Quick launch bar or use the Firefox icon on the Desktop, start the WPS 7.0 Performance Tester
 - ___ a. Click at the **WPS Performance Tester** Bookmark in Firefox, the Servlet to control the application opens
(<http://localhost:9080/PerformanceTesterWeb/Main>)



WPS 7.0 Performance Tester (An IBM Accelerated Value Example Application)

This is the frontend servlet to trigger the WPS **SenderModule** calling the WPS **ReceiverModule** on different Binding Types.

Loop Value

Binding

Object Size

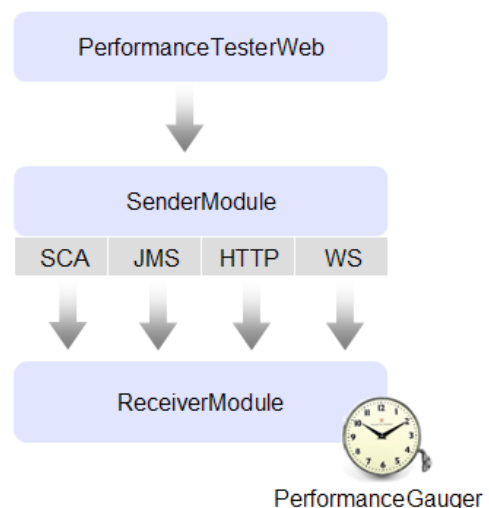
This application gives you the opportunity to run a set of very simple, configurable performance test cases. The **Loop Value** configures the number of times the **SenderModule** calls the **ReceiverModule** and transfers the Business Object. This parameter gives you the opportunity to simulate load in terms of parallel invokes (or parallel users).

The **Binding** dropdown box lets you choose between the following Binding types: **SCA sync**, **SCA async**, **JMS async**, **HTTP sync** and **WS sync**. This parameter will allow you to identify the performance differences of the mentioned Binding types.

The **Object size** lets you select between 1KB, 100KB and 1MB Objects which are added to the Business Object which are sent from the SenderModule to the ReceiverModule. This parameter simulates load in terms of Memory usage.

The PerformanceGauger collects statistical information on the Test Scenarios you run and sends this information back to you in the Browser.

When asked below, use the **Submit** Button to start a Test Scenario.



___ 3. Use Wintail to monitor the WPS SystemOut Log



Before starting the first Test Scenario its recommended to use Wintail (available on the Desktop or via the Windows quick launch bar) and open the WPS SystemOut log file with it. This utility helps you to monitor the Log output of WPS in real-time, which gives you a better understanding on what WPS is doing during potential wait times.

It already points to the file: **C:\WID7_WTE\runtimes\bi_v7\profiles\qwp\logs\server1\SystemOut.log**

Move Wintail to the bottom at your Desktop and put Firefox on top of it, which will then allow you to navigate in the Browser and monitor the SystemOut log in parallel.

___ 4. For the first time, simply take the defaults and run a Performance Test with 200 loops, SCA sync Binding and 1KB Object Size. Press Submit

___ a. The very first time you are doing this after a fresh WPS restart it might take a few moments until WPS warms up (in fact WPS fills internal caches)

___ b. Monitor Wintail and the Browser, you'll the following output:
(Note: The dates and values you will see will not exactly match the values shown below)

WPS 7.0 Performance Tester (An IBM Accelerated Value Example Application)

This is the frontend servlet to trigger the WPS **SenderModule** calling the WPS **ReceiverModule** on different Binding Types.

Loop Value	<input type="text" value="200"/>
Binding	<input type="text" value="SCA sync"/>
Object Size	<input type="text" value="1KB"/>
<input type="button" value="Submit"/>	

```
[Fri Mar 04 05:51:58 PST 2011] Initializing and starting Performance Test (with SCA sync Binding and 1KB object size)...
[Fri Mar 04 05:52:05 PST 2011] Performance Test started
[Fri Mar 04 05:52:08 PST 2011] runs 10, throughput average 3.1 ms per run, received objects size = 0.0 MB, thread name 'Default : 0', total heap 384.0 MB, free heap 154.0 MB
[Fri Mar 04 05:52:08 PST 2011] runs 20, throughput average 2.35 ms per run, received objects size = 0.0 MB, thread name 'Default : 0', total heap 384.0 MB, free heap 153.0 MB
[Fri Mar 04 05:52:08 PST 2011] runs 30, throughput average 2.0666666 ms per run, received objects size = 0.0 MB, thread name 'Default : 0', total heap 384.0 MB, free heap 153.0 MB
[Fri Mar 04 05:52:08 PST 2011] runs 40, throughput average 1.925 ms per run, received objects size = 0.0 MB, thread name 'Default : 0', total heap 384.0 MB, free heap 152.0 MB
[Fri Mar 04 05:52:08 PST 2011] runs 50, throughput average 1.86 ms per run, received objects size = 0.0 MB, thread name 'Default : 0', total heap 384.0 MB, free heap 151.0 MB
[Fri Mar 04 05:52:08 PST 2011] runs 60, throughput average 1.55 ms per run, received objects size = 0.0 MB, thread name 'Default : 0', total heap 384.0 MB, free heap 150.0 MB
[Fri Mar 04 05:52:08 PST 2011] runs 70, throughput average 1.5428572 ms per run, received objects size = 0.0 MB, thread name 'Default : 0', total heap 384.0 MB, free heap 149.0 MB
[Fri Mar 04 05:52:08 PST 2011] runs 80, throughput average 1.55 ms per run, received objects size = 0.0 MB, thread name 'Default : 0', total heap 384.0 MB, free heap 148.0 MB
[Fri Mar 04 05:52:08 PST 2011] runs 90, throughput average 1.5555556 ms per run, received objects size = 0.0 MB, thread name 'Default : 0', total heap 384.0 MB, free heap 147.0 MB
[Fri Mar 04 05:52:08 PST 2011] runs 100, throughput average 1.4 ms per run, received objects size = 0.0 MB, thread name 'Default : 0', total heap 384.0 MB, free heap 146.0 MB
[Fri Mar 04 05:52:08 PST 2011] runs 110, throughput average 1.2727273 ms per run, received objects size = 0.0 MB, thread name 'Default : 0', total heap 384.0 MB, free heap 146.0 MB
[Fri Mar 04 05:52:08 PST 2011] runs 120, throughput average 1.2916666 ms per run, received objects size = 0.0 MB, thread name 'Default : 0', total heap 384.0 MB, free heap 145.0 MB
[Fri Mar 04 05:52:08 PST 2011] runs 130, throughput average 1.1923077 ms per run, received objects size = 0.0 MB, thread name 'Default : 0', total heap 384.0 MB, free heap 144.0 MB
[Fri Mar 04 05:52:08 PST 2011] runs 140, throughput average 1.2214285 ms per run, received objects size = 0.0 MB, thread name 'Default : 0', total heap 384.0 MB, free heap 143.0 MB
[Fri Mar 04 05:52:08 PST 2011] runs 150, throughput average 1.14 ms per run, received objects size = 0.0 MB, thread name 'Default : 0', total heap 384.0 MB, free heap 142.0 MB
[Fri Mar 04 05:52:08 PST 2011] runs 160, throughput average 1.06875 ms per run, received objects size = 0.0 MB, thread name 'Default : 0', total heap 384.0 MB, free heap 141.0 MB
[Fri Mar 04 05:52:08 PST 2011] runs 170, throughput average 1.0058824 ms per run, received objects size = 0.0 MB, thread name 'Default : 0', total heap 384.0 MB, free heap 140.0 MB
[Fri Mar 04 05:52:08 PST 2011] runs 180, throughput average 0.95 ms per run, received objects size = 0.0 MB, thread name 'Default : 0', total heap 384.0 MB, free heap 139.0 MB
[Fri Mar 04 05:52:08 PST 2011] runs 190, throughput average 0.9 ms per run, received objects size = 0.0 MB, thread name 'Default : 0', total heap 384.0 MB, free heap 138.0 MB
[Fri Mar 04 05:52:08 PST 2011] runs 200, throughput average 0.935 ms per run, received objects size = 0.0 MB, thread name 'Default : 0', total heap 384.0 MB, free heap 138.0 MB
done
[Fri Mar 04 05:52:09 PST 2011] Performance Test finished
```

The Statistical outputs you're able to see in the Browser give us the following information:

- runs**, this is the number of invokes from SenderModule to ReceiverModule
- throughput average**, the cumulated average time in ms a single run takes
- object size**, this is the cumulated amount of Megabytes received
- thread name**, name and index of the current Thread pools which is used
- total heap**, the number of megabytes allocated for the heap currently
- free heap**, currently free heap of the total heap



Note

You will find **Test Case Log pages** at the end of the exercise. These pages will give you the opportunity to note any performance test results for later reference.

- ___ 5. Observe the throughput - the actual average value of the whole test run is the last row in the output. Write it down in the Test Case Log for later reference.
- ___ 6. Run another test or two with the same settings. You'll notice that the throughput significantly goes up after WPS is warmed up (e.g. from average of ~3ms for the first run to 1.25ms for the second). Write it down in the Test Case Log for later reference.



Note

You'll notice that the current Thread Pool is always **'Default'**. We are going to change this later, but for now its important to note that all the test you will run, are not only depended of the parameters in the Web UI, but also on the configuration of the WAS infrastructure underneath WPS.

- ___ 7. Try all the available binding types with the 1KB and 100KB Object sizes twice (the first run is for WPS warm-up) and notice the throughput in the **Test Case Log**.

Attention: Please **DO NOT** use the **1MB** object size yet, as this will much likely overload the 384MB Java Heap and results in running into **out of memory**.



Note

In our example the performance for the 100KB Object was as shown below:

! **SCA sync 1.4ms, SCA async 1400ms, JMS async 70ms, HTTP sync 27ms and WS sync 77ms.**

! This shows very clearly the main difference between SCA sync and async processing, but also advantages for HTTP bindings versus WS bindings or JMS bindings.

It is important to understand that any of these binding types has advantages and disadvantages for certain user scenarios, but for this simple invoke use case (like for many other use cases in the real world) the SCA sync binding is the most efficient and best performing binding – but only, if you don't need asynchronous processing.

End of exercise

Exercise review and wrap-up

Throughput is lower when using asynchronous SCA bindings as compared to synchronous SCA bindings because asynchronous SCA bindings have to perform the following additional functions:

- Serialize the incoming business object
- Write the serialized business object to a queue
- Wake up an MDB
- Read the serialized business object from a queue
- Correlate the business object and operation to a particular binding
- Spawn a thread to run the downstream module and components

As a direct contrast, when synchronous SCA bindings are used in the same JVM, the associated business object is not written to a queue and thus does not have to be serialized. Also, the downstream SCA component (or module) runs on the thread of the caller so no thread switch occurs. Whereas serialization is not directly observable, the threading behavior of both synchronous SCA same JVM and asynchronous SCA same JVM were observed in this part of the lab.

In the case of synchronous SCA bindings when binding to a component or module in the same JVM, the throughput performance of the binding operation is comparable to a direct Java-to-Java call. In fact, SCA components are implemented as EJBs and WebSphere Process Server leverages the performance optimization provided by base WebSphere Application Server when EJBs communicate with each other in the same JVM.

Exercise 2 – Configure WebSphere Application Server artifacts to tune WPS performance

What this exercise is about

This purpose of this lab is to get used to the different artifacts in WebSphere Application Server which are closely related to WPS performance. This exercise covers how to use the WebSphere administrative console to configure concurrency in the five types of WebSphere Process Server inter-module bindings.

What you should be able to do

At the end of the exercise, you should be able to:

- Create new thread pools using WAS admin console
- Configure the resource Adapter to use the new thread pools
- Tune the Object Request Broker (ORB) for inter-module synchronous SCA, cross-JVM communications (optional)
- Tune the target module for asynchronous SCA
- Tune the target modules for JMS bindings (optional)
- Tune the server for HTTP and Web Service Bindings (optional)

Introduction

This lab is designed to illustrate the throughput performance difference between synchronous SCA bindings and asynchronous SCA bindings when running in a single JVM environment and options to influence throughput.

The performance results from this lab should not be considered representative of the performance of WebSphere Process Server and should not be generally published as a general communication of WPS performance capabilities.

Exercise instructions

Part 1: Create two new Thread Pools

In this part of the exercise, you are going to create two new thread pools: **Pool_SIBus_SCA** and **Pool_SIBus_JMS**. This separates the thread pool used by the SCA runtime, and JMS bindings. The benefit of doing this allows you to easily detect which thread is spawned by which thread pool. When you examine the outputs in the Web UI or in the SystemOut log, the thread names would have either **Pool_SIBus_SCA**, **Pool_SIBus_JMS** or **WebContainer** depending on which Binding type you have triggered.

The main purpose in the real world of separating the SCA and JMS Thread Pools of course is the option to configure and maintain the Thread Pools for both Binding Types independently. A massive load on the JMS Bindings with a limited JMS Thread Pools will have less impact to the load via SCA Bindings, for example.

- ___ 1. Run a quick test in the Performance Application and inspect the Thread names for SCA sync, SCA async and JMS Binding. Note that always the Thread Pool **'Default'** is used

- ___ 2. Create two new thread pools: one for asynchronous SCA bindings, **Pool_SIBus_SCA**, and one for JMS exports, **Pool_SIBus_JMS**
 - ___ a. Create a new Tab in Firefox and select the **WPS Admin Console** bookmark (<http://localhost:9060/ibm/console/>)
 - ___ b. Click **Log in** without giving a username (security is disabled)
 - ___ c. In the administrative console, select **Servers > Server Types > WebSphere application servers**
 - ___ d. Select server1
 - ___ e. Under Additional Properties, select **Thread Pools**.
 - ___ f. Click **New** and create a new Pool with the following Properties

Name:	Pool_SIBus_SCA
Minimum Size:	5
Maximum Size:	20
Thread Timeout Activity:	5000
 - ___ g. Do not select the Checkbox for the **'Allow thread allocation beyond maximum thread size'** parameter
 - ___ h. Press the **OK** Button and **Save** the Configuration

[Application servers](#) > [server1](#) > [Thread Pools](#) > **New**

Use this page to specify a thread pool for the server to use. A thread pool enables server components to reuse threads instead of creating new threads at run time. Creating new threads is typically a time and resource intensive operation.

Configuration

General Properties

* Name

Description

* Minimum Size
 threads

* Maximum Size
 threads

* Thread inactivity timeout
 milliseconds

Allow thread allocation beyond maximum thread size

The additional properties will not be available until the general properties for this item are applied or saved.

Additional Properties

Custom Properties

__ i. repeat the last steps for the second Pool with the following properties:

Name:	Pool_SIBus_JMS
Minimum Size:	5
Maximum Size:	20
Thread Timeout Activity:	5000

__ j. Do not select the Checkbox for the '**Allow thread allocation beyond maximum thread size**' parameter

__ k. Press the **OK** Button and **Save** the Configuration

You now have created two new Thread Pools:

<input type="checkbox"/>	Pool_SIBus_JMS		5	20
<input type="checkbox"/>	Pool_SIBus_SCA		5	20

Part 2: Configure the Resource Adapters to use the new Thread Pools

- __ 2. Review and configure resource adapters
 - __ a. In the administration console, expand **Resources > Resource Adapters**.
 - __ b. Select **Resource adapters**.
 - __ c. Confirm that **Scopes** is set to **All scopes**
 - __ d. Expand the Preferences, select the **Show built-in resources** check box, and then click Apply.

Resource adapters

Use this page to manage resource adapters, which provide the fundamental interface for connecting applications to an Enterprise Information System (EIS). The WebSphere(R) Relational Resource Adapter is embedded within the product to provide access to relational databases. To access another type of EIS, use this page to install a standalone resource adapter archive (RAR) file. You can configure multiple resource adapters for each installed RAR file.

Scope: =**All scopes**

Scope specifies the level at which the resource definition is visible. For detailed information on what scope is and how it works, [see the scope settings help](#)

All scopes

Preferences

Maximum rows
20

Retain filter criteria.

Show built-in resources

Apply Reset

- __ e. Select **Platform Messaging Component SPI Resource Adapter**.
- __ f. Change the **Thread pool** alias parameter from **Default** to **Pool_SIBus_SCA**

Native library path

Isolate this resource provider

Thread pool alias
Pool_SIBus_SCA

Apply OK Reset Cancel

- __ g. Click **Apply**; then **Save**

__ h. Back to the **Resource Adapters** list, select **SIB JMS Resource Adapter** at the server scope level

Select	Name	Scope
<input type="checkbox"/>	Platform Messaging Component SPI Resource Adapter	Node=qnode
<input type="checkbox"/>	SIB JMS Resource Adapter	Cell=qcell
<input type="checkbox"/>	SIB JMS Resource Adapter	Node=qnode
<input type="checkbox"/>	SIB JMS Resource Adapter	Node=qnode,Server=server1
<input type="checkbox"/>	WebSphere MQ Resource Adapter	Cell=qcell
<input type="checkbox"/>	WebSphere MQ Resource Adapter	Node=qnode
<input type="checkbox"/>	WebSphere MQ Resource Adapter	Node=qnode,Server=server1
<input type="checkbox"/>	WebSphere Relational Resource Adapter	Cell=qcell
<input type="checkbox"/>	WebSphere Relational Resource Adapter	Node=qnode
<input type="checkbox"/>	WebSphere Relational Resource Adapter	Node=qnode,Server=server1

Total 10

__ i. Change the Thread pool alias parameter here from **SIBJMSRThreadPool** to **Pool_SIBus_JMS**

Native path

Thread pool alias

Pool_SIBus_JMS

Apply OK Reset Cancel

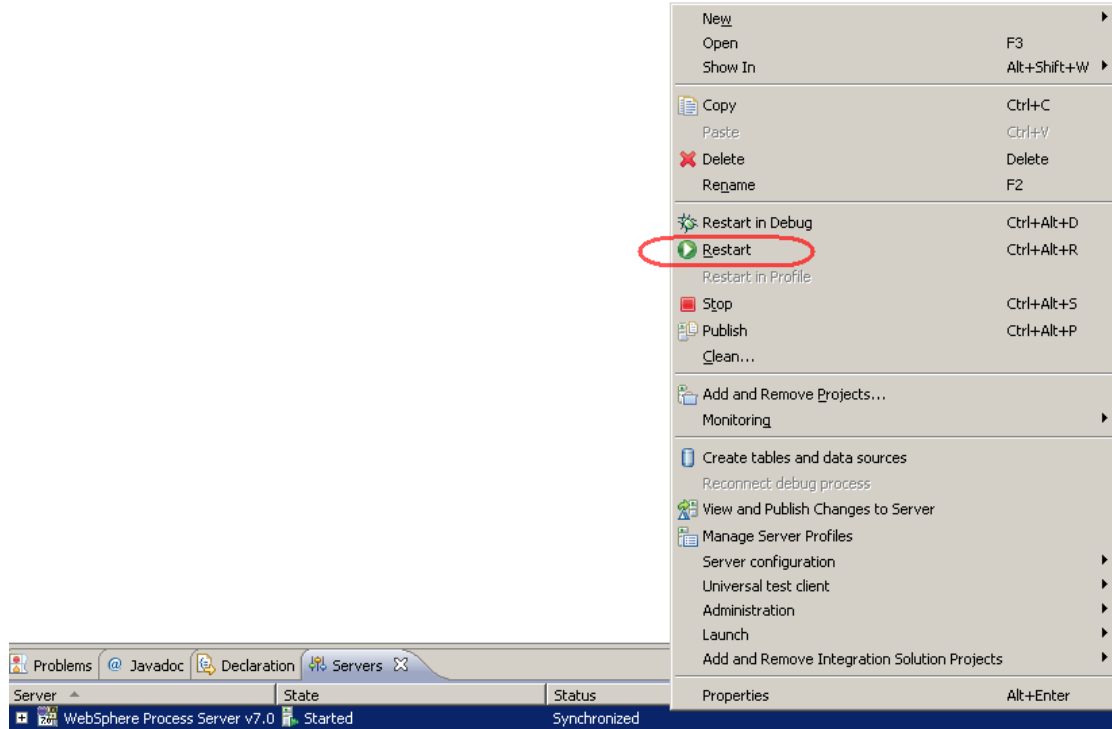
__ j. Click **Apply** and then **Save**

__ k. Optional: Repeat the configuration of the **SIB JMS Resource Adapter** at the node and cell levels as required (depending on the WebSphere Process Server deployment configuration and future usage expectations).

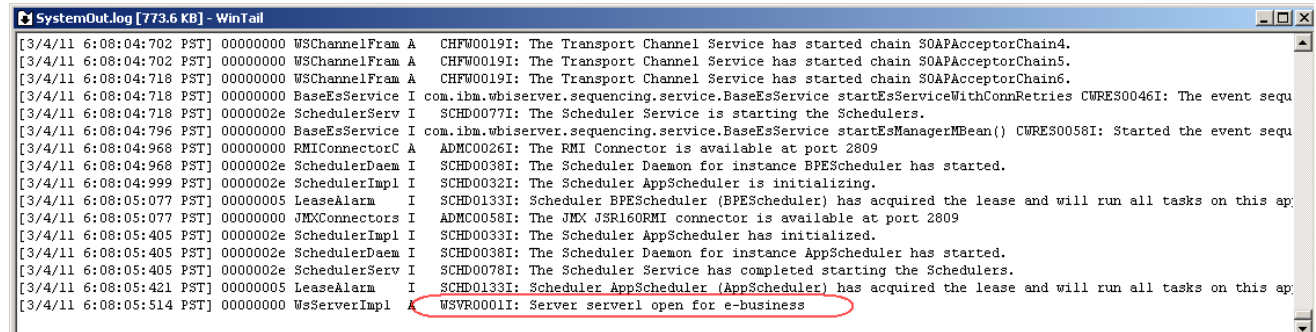
__ 3. Restart WPS to make these changes effective

__ a. in WebSphere Integration Developer, change to the Server view and right-click **WebSphere Process Server V7.0**

__ b. select **restart**



c. WebSphere Process Server now gets restarted, you can monitor the progress in the Wintail window, wait until you see the message:
Server server1 open for e-business



d. Use the Performance Tester application again and run a test case for SCA sync and JMS async, inspect the Thread Name in the output log. You will notice that both bindings now use separate Thread Pools.
Notice the throughput in the **Test Case Log**.



Information

Note that these two internal adapters were sharing the same thread pool, Default, in the initial configuration. You created two new thread pools, Pool_SIBus_JMS and Pool_SIBus_SCA, to separate the thread pool.

Optional Part 3: Tune Object Request Broker for inter-JVM synchronous SCA

The **ReceiverModule** contains many exports, including an SCA export. The **SenderModule** uses SCA to call the **ReceiverModule**. If the **SenderModule** module is using SCA synchronous and if it is deployed in a different JVM than the **ReceiverModule** module, Remote Method Invocation (RMI) will be used between the modules and the ORB needs to be configured for optimal concurrency in the **ReceiverModule** module.



Note

In this example both modules are deployed to one and the same JVM, which means that the **SenderModule** calls the **ReceiverModule** via a direct EJB reference. The ORB is not used in this scenario. However in a real production deployment, you may want to install Modules on different JVMs which make it necessary to know how to tune the ORB.

- ___ 3. Observe the threading configuration of the ORB.
- ___ a. Open Firefox and select the **WPS Admin Console** bookmark (<http://localhost:9060/ibm/console/>)
- ___ b. Click **Log in** without giving a username (security is disabled)
- ___ c. From the left, expand **Servers > Server Types > WebSphere Application Servers**.
- ___ d. Select server1.
- ___ e. Under **Additional Properties**, select Thread Pools
- ___ f. Select **ORB.Thread.Pool** from the list
- ___ g The values should be set as follows:

Minimum Size:	10
Maximum Size:	50
Thread Timeout Activity:	3500
- ___ h. There are four parameters on the page, and notice that the maximum thread count (**Maximum Size**) is set to **50** by default. The ORB thread is a shared resource. Therefore, you must consider all other applications that are requesting threads from this thread pool, and tune this value accordingly

[Application servers](#) > [server1](#) > [Thread Pools](#) > [ORB.thread.pool](#)

Use this page to specify a thread pool for the server to use. A thread pool enables server components to reuse threads instead of creating new threads at run time. Creating new threads is typically a time and resource intensive operation.

Configuration

General Properties

* Name:

Description:

* Minimum Size: threads

* Maximum Size: threads

* Thread inactivity timeout: milliseconds

Allow thread allocation beyond maximum thread size

Additional Properties

- [Custom Properties](#)

 **Note**

Some benchmarks achieve better performance with limited sized ORB thread pools. Therefore, a reasonably sized thread pool may be best and the “beyond maximum” check box should not be selected. Additionally, some benchmarks benefit from increasing the “Thread inactivity timeout.”

The main purpose of this portion of the lab was to illustrate how to configure the ORB when using SCA synchronous between modules in different JVMs.

Part 4: Tune the target module for asynchronous SCA

For the SCA asynchronous invoke of the **ReceiverModule**, a **J2C Activation Specification** is used to configure the corresponding SCA export. The main component on WAS infrastructure level therefore is the **Platform Messaging Component SPI Resource Adapter**, which maintains the mentioned **J2C Activation Specification** for the **ReceiverModule**.

- ___ 1. Observe the threading configuration of the ReceiverModule
- ___ a. From the left menu, expand **Resources > Resource Adapters > Resource adapters** in the administrative console.
- ___ b. Select **Platform Messaging Component SPI Resource Adapter**
- ___ c. Under Additional Properties, select **J2C activation specifications** and click **ReceiverModule_AS**.
- ___ d. Under Additional Properties on the right, select **J2C activation specification custom properties**
- ___ e. Observe the table of custom properties. Find the **maxConcurrency** parameter. This parameter is set to 10 by default.

[Resource adapters](#) > [Platform Messaging Component SPI Resource Adapter](#) > [J2C activation specifications](#) > [ReceiverModule_AS](#) > **Custom properties**

Use this page to specify custom properties that your enterprise information system (EIS) requires for the resource providers and resource factories that you configure. For example, most database vendors require additional custom properties for data sources that access the database.

⊕ Preferences

Name	Value	Description	Required
destinationType	Queue		true
subscriptionName			false
useServerSubject	true		false
messageSelector	(SI_CorrelationID IS NULL) AND (SI_ExceptionReason IS NULL) AND (user.scaStoredMsgId IS NULL)		false
targetType			false
maxConcurrency	10		false



Information

The maximum number of instances of each message-driven bean is controlled by the **maxConcurrency** setting in the activation specification in the SPI Resource Adapter. In fact it means the number of Threads which are allowed to be spawned to manage the MDBs. This maximum concurrency limit helps prevent a temporary build up of messages from starting an excessive number of MDB instances. By default, the maximum number of concurrent MDB instances is set to 10.

Changing the number of parallel MDBs for a receiver has a significant impact on parallel asynchronous SCA processing especially in heavy load scenarios.

Setting the **maxConcurrency** to 1 in fact means event sequencing, which might be necessary if you intend to keep the order of the incoming SCA calls as they have been invoked by the sender.

__ 2. Change the **maxConcurrency** settings

__ a. Click the **maxConcurrency** link

__ b. change the **maxConcurrency** value to 1

The screenshot shows a configuration form with the following fields and controls:

- Name:** maxConcurrency
- Value:** 1 (circled in red)
- Description:** (empty text area)
- Type:** java.lang.Integer (dropdown menu)
- Buttons:** Apply, OK, Reset, Cancel

__ c. **Apply** and **Save** the changes

__ 3. Restart the **SenderModuleApp** and **ReceiverModuleApp**

__ a. Still in the WAS admin console, in the left navigator select **Applications > Application Types > WebSphere enterprise applications**


__ b. (optional) Filter for *Module*

Enterprise Applications


Use this page to manage installed applications. A single application can be deployed onto multiple servers.

☒ Preferences

- ___ c. Select **SenderModuleApp** and **ReceiverModuleApp**, press the **Stop** Button
- ___ d. Select **SenderModuleApp** and **ReceiverModuleApp**, press the **Start** Button
- ___ 4. Since we now have the **maxConcurrency** set to 1, run the Performance Tester again and select SCA async Binding, 200 loops and Object Size 1KB and 100KB
Notice the throughput in the **Test Case Log**.
- ___ 5. Go back to step 2. and change the **maxConcurrency** to 100
- ___ 6. Since we now have the **maxConcurrency** set to 100, run the Performance Tester again and select SCA async Binding, 200 loops and Object Size 1KB and 100KB
Notice the throughput in the **Test Case Log**.

 **Note**

Since the SenderModuleApp and ReceiverModuleApp needs to be restarted to pickup the changed Activation Specification setting, the WPS internal caches needs to be refreshed. It is therefore recommended to run at least one test prior the actual one you are going to notice.

 **Information**

Changing the number of **maxConcurrency** in fact changes the number of parallel MDBs for a receiver. This has a significant impact on parallel asynchronous SCA processing especially in heavy load scenarios.

You may have noticed just slightly different throughputs by having **maxConcurrency** at 1 compared to 100. This has to do with the nature of the Performance Tester application, which basically represents one single client. If you want to take full advantage of a higher concurrency, open several Firefox Tabs with Performance tester and run parallel scenarios.

Optional Part 5: Tune the target modules for JMS bindings

- ___ 1. Observe the threading configuration for a JMS binding
 - ___ a. In the **administrative console** pane, expand **Resources > JMS > JMS Providers**, and select **Default messaging provider** on the server level

Select	Name	Description	Scope
You can administer the following resources:			
	Default messaging provider	Default messaging provider	Cell=qcell
	Default messaging provider	Default messaging provider	Node=qnode
	Default messaging provider	Default messaging provider	Node=qnode,Server=server1
	V5 default messaging provider	V5 Default Messaging Provider	Node=qnode,Server=server1
	V5 default messaging provider	V5 Default Messaging Provider	Node=qnode
	V5 default messaging provider	V5 Default Messaging Provider	Cell=qcell
	WebSphere MQ messaging provider	WebSphere MQ Messaging Provider	Node=qnode,Server=server1
	WebSphere MQ messaging provider	WebSphere MQ Messaging Provider	Node=qnode
	WebSphere MQ messaging provider	WebSphere MQ Messaging Provider	Cell=qcell
Total 9			

- ___ b. Under **Additional Properties** on the right, select **Activation specification**
- ___ c. Select **ReceiverModule.JMS_Export_AS** and view its properties.
- ___ d. Scroll down the page and find the **Additional** section of parameters. There are two parameters in the Additional section

Maximum batch size, Maximum concurrent MDB invocations per endpoint

Additional

Maximum batch size

Maximum concurrent MDB invocations per endpoint

i Information

The maximum number of instances of each message-driven bean is controlled by the **Maximum concurrent MDB invocations per endpoint** setting in the activation specification in the JMS Resource Adapter. In fact it means the number of Threads which are allowed to be spawned to manage the MDBs. The meaning of this parameter is the same as it was the maxConcurrency for the SPI Resource Adapter.

Changing the number of parallel MDBs for a receiver has a significant impact on parallel asynchronous JMS processing especially in heavy load scenarios.

Setting the **Maximum concurrent MDB invocations per endpoint** to 1 in fact means event sequencing, which might be necessary if you intend to keep the order of the incoming JMS calls as they have been invoked by the sender.

__ 2. Change the **Maximum concurrent MDB invocations per endpoint** settings

__ a. change the value for **Maximum concurrent MDB invocations per endpoint** value to 1

Additional

Maximum batch size
1

Maximum concurrent MDB invocations per endpoint
1

__ b. **Apply** and **Save** the changes

__ 3. Restart the **SenderModuleApp** and **ReceiverModuleApp**

__ a. Still in the WAS admin console, in the left navigator select **Applications > Enterprise Applications**

__ b. (optional) Filter for *Module*

Enterprise Applications

Use this page to manage installed applications. A single application can be deployed onto multiple servers.

⊞ Preferences

<input type="button" value="Start"/> <input type="button" value="Stop"/> <input type="button" value="Install"/> <input type="button" value="Uninstall"/> <input type="button" value="Update"/> <input type="button" value="Rollout Update"/> <input type="button" value="Remove File"/> <input type="button" value="Export"/> <input type="button" value="Export DDL"/> <input type="button" value="Export File"/>		
Select	Name <input type="text"/>	Application Status <input type="button" value="Refresh"/>
	Filter: *Module*	
To filter the following table, select the column by which to filter, then enter filter criteria (wildcards: *,?,%). Filter: <input type="text"/> Search term(s): <input type="text"/> <input type="button" value="Go"/>		
<input checked="" type="checkbox"/>	ReceiverModuleApp	<input type="button" value="Refresh"/>
<input checked="" type="checkbox"/>	SenderModuleApp	<input type="button" value="Refresh"/>
Total 21 Filtered total: 2		

- ___c. Select **SenderModuleApp** and **ReceiverModuleApp**, press the **Stop** Button
- ___d. Select **SenderModuleApp** and **ReceiverModuleApp**, press the **Start** Button
- ___ 4. Run the Performance Test for the **Maximum concurrent MDB invocations per endpoint** to be 1, select JMS async Binding, 200 loops and Object Size 1KB and 100KB
Notice the throughput in the **Test Case Log**.
- ___ 5. Redo the last steps since 2. and change the **Maximum concurrent MDB invocations per endpoint** to 100
- ___ 6. Run the Performance Test for the **Maximum concurrent MDB invocations per endpoint** to be 100, select JMS async Binding, 200 loops and Object Size 1KB and 100KB
Notice the throughput in the **Test Case Log**.

**Note**

Since the SenderModuleApp and ReceiverModuleApp needs to be restarted to pickup the changed Activation Specification setting, the WPS internal caches needs to be refreshed. It is therefore recommended to run at least one test prior the actual one you are going to notice.

Optional Part 6: Tune the server for HTTP and Web Service Bindings

When you have run the Performance Tester for the Binding Types **HTTP sync** or **WS sync** you will have noticed that the default Thread name is always **WebContainer**. This is the Default Thread Pool for any Web Container related services and therefore has an important impact on throughput and performance of those two Binding types.

- __ 1. Observe the threading configuration of the Application Servers Web Container
 - __ a. Open Firefox and select the **WPS Admin Console** bookmark (<http://localhost:9060/ibm/console/>)
 - __ b. Click **Log in** without giving a username (security is disabled)
 - __ c. From the left, expand **Servers > Server Types >** and select **WebSphere application servers**.
 - __ d. Select server1.
 - __ e. Under **Additional Properties**, select Thread Pools
 - __ f. Then, select **WebContainer**. Verify that the following page is displayed:

[Application servers](#) > [server1](#) > [Thread pools](#) > **WebContainer**

Use this page to specify a thread pool for the server to use. A thread pool enables server components to reuse threads instead of creating new threads at run time. Creating new threads is typically a time and resource intensive operation.

Configuration

General Properties	Additional Properties
<p>* Name <input type="text" value="WebContainer"/></p> <p>Description <input type="text"/></p> <p>* Minimum Size <input type="text" value="50"/> threads</p> <p>* Maximum Size <input type="text" value="50"/> threads</p> <p>* Thread inactivity timeout <input type="text" value="60000"/> milliseconds</p> <p><input type="checkbox"/> Allow thread allocation beyond maximum thread size</p> <p><input type="button" value="Apply"/> <input type="button" value="OK"/> <input type="button" value="Reset"/> <input type="button" value="Cancel"/></p>	<p>■ Custom properties</p>



Note

The **Allow thread allocation beyond maximum thread size** option allows the WebContainer Thread Pool to grow without a limit. The WebContainer thread pool is a shared resource. Consider selecting the **Allow thread allocation beyond maximum thread size** check box to ensure that a sufficient number of threads are available from this thread pool.

Caution: Running with unlimited threads may cause memory issues due to the native, per thread address space required, depending on the load on the Web Container.

__ 2. Change the **Minimum Size** and **Maximum size** of the WebContainer Thread Pool

__ a. change the value for **Minimum Size** value to 1 and the **Maximum size** to 2

General Properties

* Name
WebContainer

Description

* Minimum Size
1 threads

* Maximum Size
2 threads

* Thread inactivity timeout
60000 milliseconds

Allow thread allocation beyond maximum thread size

Apply OK Reset Cancel

__ c. **Apply** and **Save** the changes

__ 3. Restart the **SenderModuleApp** and **ReceiverModuleApp**

__ 4. Run the Performance Test for the Minimum Size to be 1, Maximum Size to be 2, try **HTTP sync** and **WS sync** Bindings, 200 loops and Object Size 1KB and 100KB Notice the throughput in the **Test Case Log**.

__ 5. Redo the last steps since 2. and change the **Minimum Size** to be 10, **Maximum Size** to be 50

__ 6. Run the Performance Test for the Minimum Size to be 10, Maximum Size to be 50, try **HTTP sync** and **WS sync** Bindings, 200 loops and Object Size 1KB and 100KB Notice the throughput in the **Test Case Log**.

End of exercise

Exercise review and wrap-up

In this exercise you have changed parameters in the WebSphere Application Server admin console which have can have important influence on performance of the different WPS Binding types.

You have created dedicated Thread Pools and separated the SCA and JMS Thread Management by configuring the Resource Adapters to use these Thread Pools. You now know how to tune the ORB in case you have Inter-Module synchronous SCA communication between several JVMs. You have tuned the asynchronous SCA and JMS Binding types by changing the maximum concurrency of the MDBs which manage the SCA exports. And you have tuned the synchronous HTTP and Web Services Binding (SOAP1.1/HTTP via JAX/RPC) by configuring the WebContainer's default Thread Pool.

Exercise 3 – WebSphere Process Server Memory tuning

What this exercise is about

During this exercise you are going to use the Performance Tester WPS Modules to again measure performance on different WPS Binding styles. This time the purpose is to observe the Java Heap statistics for a WebSphere Process Server based Solution by collecting and graphing the JVM verbose garbage collection (verbose:gc) output.

What you should be able to do

At the end of the exercise, you should be able to:

- Analyze java heap statistics for the **SenderModule** and **ReceiverModule** applications running in WPS by collecting verbose:gc output for small, medium and large objects, massive load and out of memory conditions
- Use the IBM Pattern Modeling and Analysis Tool to graph verbose:gc trace log output

Introduction

The purpose of this lab is to observe Java heap statistics for a WPS based solution by collecting and graphing verbose:gc logs.

Exercise instructions

In this exercise, you are going to run some tests to collect verbose garbage collector (GC) traces and analyze the data using the **IBM Pattern Modeling and Analysis Tool (PMAT)**, which is a free download from the IBM alpha works page:

<http://www.alphaworks.ibm.com/tech/pmat>

Part 1: Collecting Java heap memory statistics with SCA sync binding

The purpose of this part is to establish a baseline Java heap behavior graph.

- __ 1. Clear the currently existing native_stderr.log and run the first Test case
 - __ a. In the WebSphere Integration Developers **Server** view, right-click the WebSphere Process Server 7.0 and select **Stop**. Wait until the server was shut down.
 - __ b. open Windows Explorer via the Windows quick launch bar, navigate to C:\WID7_WTE\runtimes\bi_v7\profiles\qwpes\logs\server1
 - __ c. rename the file native_stderr.log to **orig_native_stderr.log**
 - __ d. In WID, **start** the WPS server again. Wait until the server is started.



Note

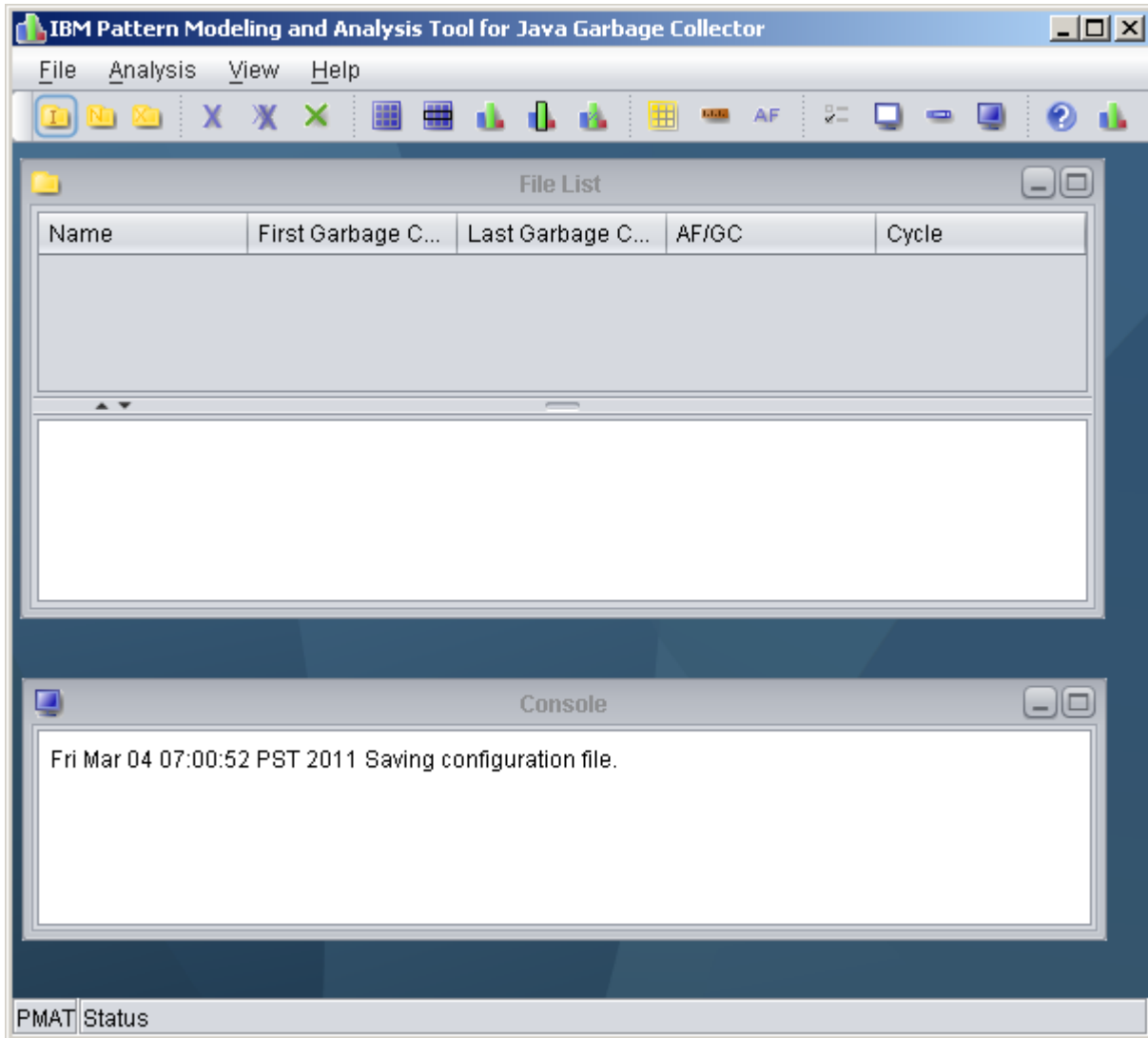
Note that the WebSphere Process Server is currently configured to have a Maximum Heap size of 384MB.

- __ e. Open Firefox and click the **Performance Tester** bookmark (<http://localhost:9080/PerformanceTesterWeb/Main>)
- __ f. Run the first Performance Test three times in summary and write down the **current timestamp** of each run for later reference. **Wait approximately one minute between the runs.**

Use the following values for the Performance test.

Loop Value:	200
Binding:	SCA sync
Object Size:	1MB

- __ 2. Use the PMAT Tool to create a graph to analyze this test result
 - __ a. In the WebSphere Integration Developers **Server** view, right-click the WebSphere Process Server 7.0 and select **Stop**. Wait until the server was shut down.
 - __ b. rename the file native_stderr.log to **test1.log**
 - __ c. Create a new directory C:\MemoryTests and copy the **test1** into this directory
 - __ d. On the Desktop double-click the **IBM Pattern Modeling and Analysis Tool** shortcut



Information

The **IBM Pattern Modeling and Analysis Tool** allows you to analyze garbage collection activities in a graphical way.

- ___ e. Select **File > Open verbose:gc Files (IBM SDK)**
 - ___ f. Navigate to C:\MemoryTests\test1.log and click **Open**. Analysis is displayed once processing is completed. Statistics of verbose:gc data are displayed as well as the analysis of each error.
- **File name** : Location and file name of verbose:gc trace
 - **Number of verboseGC cycles** : Number of JVM restarts
 - **Number of Garbage Collections** : GC frequency

- **Number of Allocation failures** : Allocation Failure frequency
- **First Garbage Collection** : Timestamp of the First GC
- **Last Garbage Collection** : Timestamp of the Last GC
- **Number of Java heap exhaustion** : Number of OutOfMemory errors

File List

Name	First Garbage Collec...	Last Garbage Collec...	AF/GC	Cycle
native_stderr.log	Fri Mar 4 05:05:58 20...	Fri Mar 4 06:50:04 20...	335/374	3

- **File name** : C:\WID7_WTE\runtimes\bi_v7\profiles\qwwps\logs\server1\native_stderr.log
- **Number of verboseGC cycles** : 3
- **Number of Garbage Collections** : 374
- **Number of Allocation failures** : 335
- **First Garbage Collection** : Fri Mar 4 05:05:58 2011
- **Last Garbage Collection** : Fri Mar 4 06:50:04 2011
- **Number of Java heap exhaustion** : 0
- **Overall Garbage Collection overhead** : 1.86%
- **Maximum Garbage Collection overhead** : 100% (Fri Mar 4 05:06:11 2011)
- **Number of 100% AF overhead** : 3
- **Total Garbage Collection pause** : 115 seconds
- **Maximum Tenured Area usage** : 248,643,096 bytes (Fri Mar 4 06:50:04 2011)
- **Average Tenured Area usage** : 151,387,661 bytes
- **Number of Explicit Garbage Collection** : 39
- **Maximum Allocation Request** : 2,097,168 bytes (Fri Mar 4 06:21:27 2011)
- **There is no object request larger than 10 M bytes.**

● **Java Heap Activity Analysis and Recommendations report**

Garbage collection start / finish	Analysis	Recommendations
#1 Fri Mar 4 05:05:58 2011 Fri Mar 4 05:35:28 2011		

Configuration
gcPolicy :

PMAT Status

___ g. Select **Analysis > GC View All** to view the verbose:gc data in a table format. To sort each column, click a column header. Maximize the window to see all columns.

- **Since** : Time (in milliseconds) elapsed since last allocation failure
- **Freed** : Size (in bytes) of space that was freed during garbage collection
- **Needed** : Size (in bytes) of space that was requested during allocation failure
- **Free** : Size (in bytes) of Java heap after garbage collection
- **Total** : Size (in bytes) of Java heap after garbage collection
- **Completed** : Time (in milliseconds) spent during allocation failure
- **GC Completed**: Time (in milliseconds) spent during garbage collection
- **Overhead**: Time (as a percentage) spent in allocation failure
- **Mark**: Time (in milliseconds) spent in mark phase
- **Sweep**: Time (in milliseconds) spent in sweep phase
- **Compact**: Time (in milliseconds) spent in compact phase
- **Exhausted**: Whether there was insufficient space to satisfy allocation failure

Used Tenure...	Free Tenure...	Total Tenure...	Needed	Free Tenure...	Total Tenured...	Free Nurse...	Total Nurse...	Free...	Total...	AF C...	Since	Mark	Sweep	Co...	GC C...
638,784	301,351,104	301,989,888	2,208	301,351,104	301,989,888	42,190,848	50,331,648	0	50,3...	45	0	0	0	0	43
639,768	301,350,120	301,989,888	56	301,350,120	301,989,888	41,660,112	50,331,648	0	50,3...	20	209	0	0	0	20
1,237,880	300,752,008	301,989,888	6,792	300,752,008	301,989,888	38,965,552	50,331,648	0	50,3...	31	4,158	0	0	0	31
2,162,400	299,827,488	301,989,888	48	299,827,488	301,989,888	35,445,776	50,331,648	0	50,3...	37	1,630	0	0	0	36
3,323,976	298,665,912	301,989,888	232	298,665,912	301,989,888	35,280,592	52,029,952	0	50,3...	37	950	0	0	0	37
4,224,792	297,765,096	301,989,888	24	297,765,096	301,989,888	38,628,704	57,568,768	0	52,0...	43	1,002	0	0	0	43
4,770,016	297,219,872	301,989,888	208	297,219,872	301,989,888	41,688,344	61,897,728	0	57,5...	43	2,499	0	0	0	43
13,994,520	287,995,368	301,989,888	135,704	295,758,752	301,989,888	49,956,608	65,630,208	63,1...	61,8...	59	1,240	0	0	0	59
12,425,008	289,564,880	301,989,888	0	287,919,688	301,989,888	48,534,848	65,630,208	24,1...	65,6...	58	0	52	3	0	57
14,725,560	287,264,328	301,989,888	128	288,014,632	301,989,888	48,108,088	67,966,976	0	65,6...	41	2,171	0	0	0	41
18,807,616	283,182,272	301,989,888	32	286,110,096	301,989,888	51,229,064	70,618,624	0	67,9...	96	7,011	0	0	0	96
16,407,352	285,582,536	301,989,888	0	282,323,760	301,989,888	50,874,576	70,618,624	33,8...	70,6...	412	9,480	141	19	248	411
20,475,280	281,514,608	301,989,888	32	285,147,400	301,989,888	54,553,008	72,811,008	0	70,6...	63	2,656	0	0	0	62
24,891,960	277,097,928	301,989,888	48	280,745,840	301,989,888	31,455,232	59,309,568	0	72,8...	72	1,681	0	0	0	71
24,127,640	277,862,248	301,989,888	0	276,894,760	301,989,888	29,631,536	59,309,568	12,3...	59,3...	65	3,870	62	1	0	65
23,849,160	278,140,728	301,989,888	0	277,859,736	301,989,888	30,687,656	59,309,568	29,5...	59,3...	200	1	59	1	137	200
27,685,856	274,304,032	301,989,888	32	277,881,992	301,989,888	32,475,424	61,675,008	0	59,3...	70	3,505	0	0	0	68
34,208,736	267,781,152	301,989,888	48	274,061,760	301,989,888	35,320,776	63,663,616	0	61,6...	71	1,840	0	0	0	70
39,369,880	262,620,008	301,989,888	72	267,477,912	301,989,888	37,943,264	65,532,416	0	63,6...	65	1,390	0	0	0	65
57,326,144	244,663,744	301,989,888	64	262,117,744	301,989,888	53,761,664	67,265,024	0	65,5...	110	2,078	0	0	0	109
61,550,584	240,439,304	301,989,888	48	244,424,936	301,989,888	60,257,400	71,560,192	0	67,2...	56	1,491	0	0	0	56

___ h. Select **Analysis > Graph View All**

You can click buttons to display the data of your interest. For example, you can enable the Free tenured (After), Used Tenured (After) and Overhead buttons to display the free space after the GC, memory used after GC, and GC overhead data in a graphical view as below

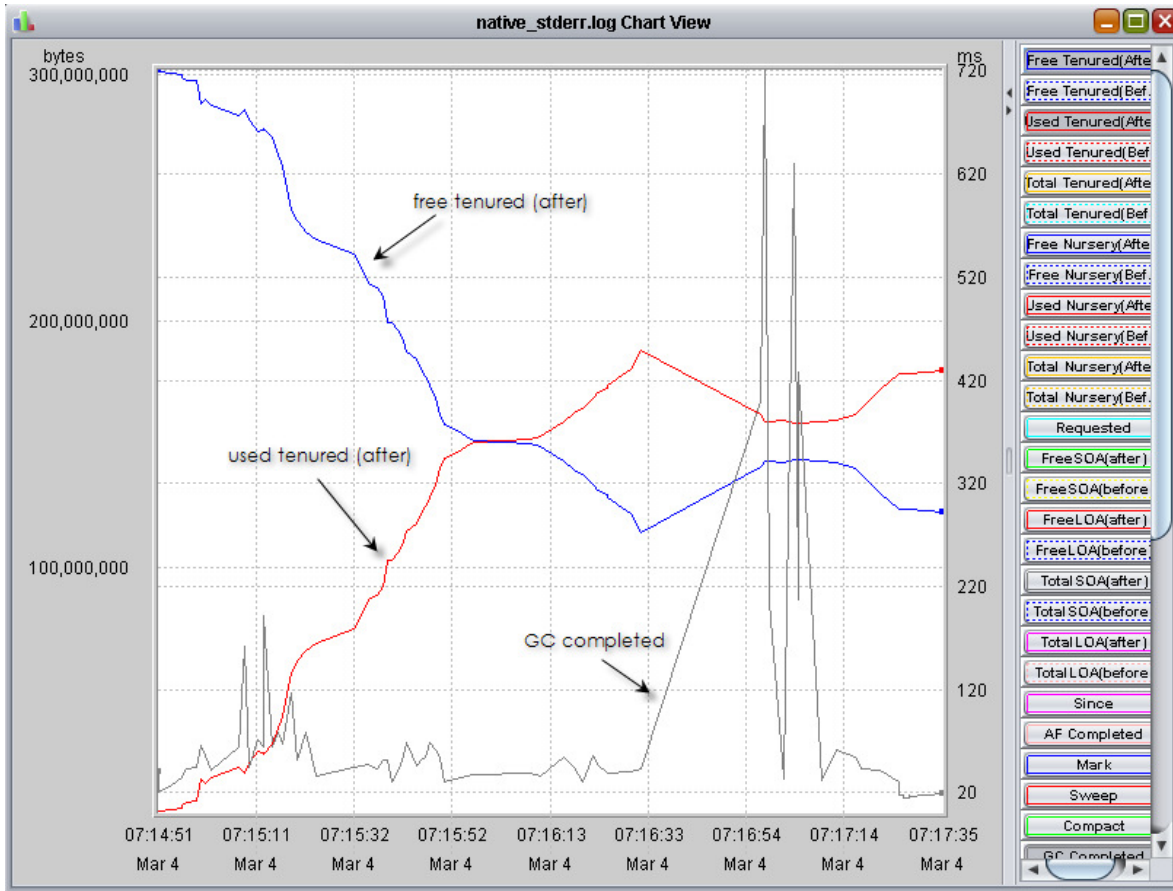


Chart Buttons descriptions are as follows:

- **Since** : Time elapsed since last allocation failure
- **Freed** : Size of space that was freed during garbage collection
- **Requested** : Size of space that was requested during allocation failure
- **Free** : Size of Java heap after garbage collection
- **Total** : Size of Java heap after garbage collection
- **Completed** : Time spent during allocation failure
- **GC Completed**: Time spent during garbage collection
- **Overhead**: Time spent in allocation failure
- **Zoom In**: Zoom in on the X-axis
- **Zoom Out**: Zoom out on the X-axis
- **Center**: Moves a point to center
- **Select**: Brings up GC view of a point



Information

In this particular test, there is no Java memory issue to be found. The graph shows reasonable balance between used and free memory space.

Part 2: Collecting Java memory statistics with SCA async binding

The purpose of this part is to observe the memory utilization for a large number of asynchronous SCA calls with big Business Objects attached.

- ___ 1. Clear the currently existing native_stderr.log and run the second Test case
- ___ a. Unless the server is not stopped already, in the WebSphere Integration Developers **Server** view, right-click the WebSphere Process Server 7.0 and select **Stop**. Wait until the server was shut down.
- ___ b. open Windows Explorer via the Windows quick launch bar, navigate to C:\WID7_WTE\runtimes\bi_v7\profiles\qwps\logs\server1
- ___ c. delete the current native_stderr.log
- ___ d. In WID, start the WPS server again. Wait until the server is started.
- ___ e. Open Firefox and click the **Performance Tester** bookmark (http://localhost:9080/PerformanceTesterWeb/Main)
- ___ f. It is useful to have Firefox and Wintail pointed to the Systemout log open. Run the Performance Test with the following values:
 - Loop Value: 400
 - Binding: **SCA async**
 - Object Size: 1MB
- ___ f. After a few moments you will notice that the throughput sinks dramatically and the >>>> **send BO messages** come slower than before. Right after that you will noticed that much likely Heapdumps and javacores are written to C:\WID7_WTE\runtimes\bi_v7\profiles\qwps. Use Windows Explorer and check this directory – if you see files starting with ,heapdump' there, your JVM ran into OutOfMemory already. Additionally you'll find many ffdc files recoding the OuOfMemory condition. You'll find those in C:\WID7_WTE\runtimes\bi_v7\profiles\qwps\logs\ffdc

```
Caused by: com.ibm.websphere.sca.ServiceRuntimeException: java.lang.OutOfMemoryError: caused by: java.lang.OutOfMemoryError
    at com.ibm.ws.sca.internal.proxy.impl.ProxyInvocationHandlerImpl.invokeAsync(ProxyInvocationHandlerImpl.java:1106)
    at com.ibm.ws.sca.internal.proxy.impl.ProxyInvocationHandlerImpl.invoke(ProxyInvocationHandlerImpl.java:723)
    at $Proxy61.invokeAsync(Unknown Source)
    at impl.SenderComponentImpl.send(SenderComponentImpl.java:168)
    ... 33 more
Caused by: java.lang.OutOfMemoryError
    at java.util.Arrays.copyOfRange(Arrays.java:4038)
    at java.util.Arrays.copyOfOf(Arrays.java:3770)
    at java.io.ByteArrayOutputStream.write(ByteArrayOutputStream.java:120)
    at sun.nio.cs.StreamEncoder$CharsetSE.writeBytes(StreamEncoder.java:343)
    at sun.nio.cs.StreamEncoder$CharsetSE.implFlushBuffer(StreamEncoder.java:413)
    at sun.nio.cs.StreamEncoder$CharsetSE.implFlush(StreamEncoder.java:417)
    at sun.nio.cs.StreamEncoder.flush(StreamEncoder.java:161)
```

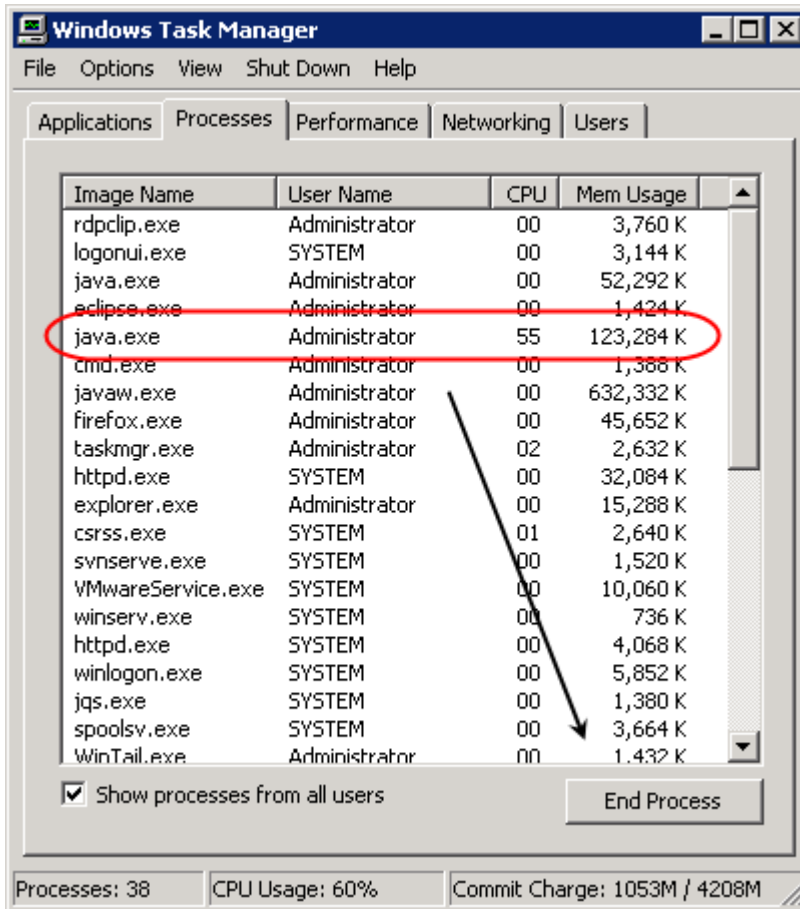
The reason for this is that we have tried to send too many Business Objects from the **SenderModule** to the **ReceiverModule** (which in fact get queued into the JMS infrastructure of WAS). With a current Heap size of 384 MB this causes the Java Heap exhaustion which ends up in the **OutOfMemory** exception.



Information

Since WPS ran into an **OutOfMemory**, the JVM might now be in an instable state, because not enough Java Heap is available to allocate Objects for several applications (which may run on the JVM in real production systems).

- ___ g. Recover to a stable state by shutting down WPS via the admin console or WID. Please close PMAT aswell. In case this fails or takes too long it is also possible to force the process to end by terminating it via the Windows Process Explorer.



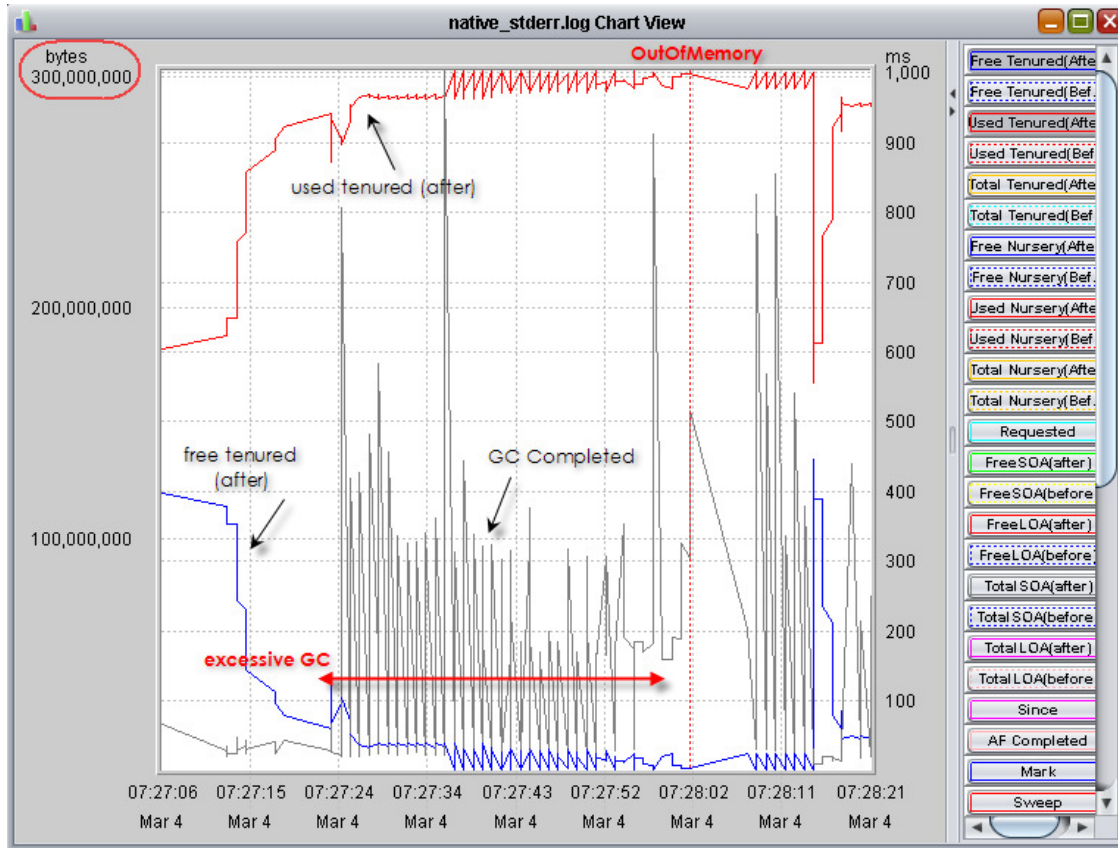
- ___ h. open Windows Explorer via the Windows quick launch bar, navigate to C:\WID7_WTE\runtimes\bi_v7\profiles\qwpw\logs\server1

- ___ i. rename the file native_stderr.log to **test2.log**

- ___ c. copy the **test2.log** into the directory C:\MemoryTests

- ___ 3. Use the PMAT Tool to create a graph to analyze this test result

- ___ a. open the file C:\MemoryTests\test2.log



Information

You will notice that between 07:27:24 and 07:27:57 the GC runs much more frequent and with longer durations. These are the desperate tries to find memory in the heap that can be freed. This condition is known as **Excessive Garbage Collection**, which is often followed by an OutOfMemory (OOM) Exception if the allocation requests proceed. In this example the OOM occurs at around 07:28:00.

Since WPS will run into the same issue after a restart, change the heap to a more appropriate size, like 1024MB

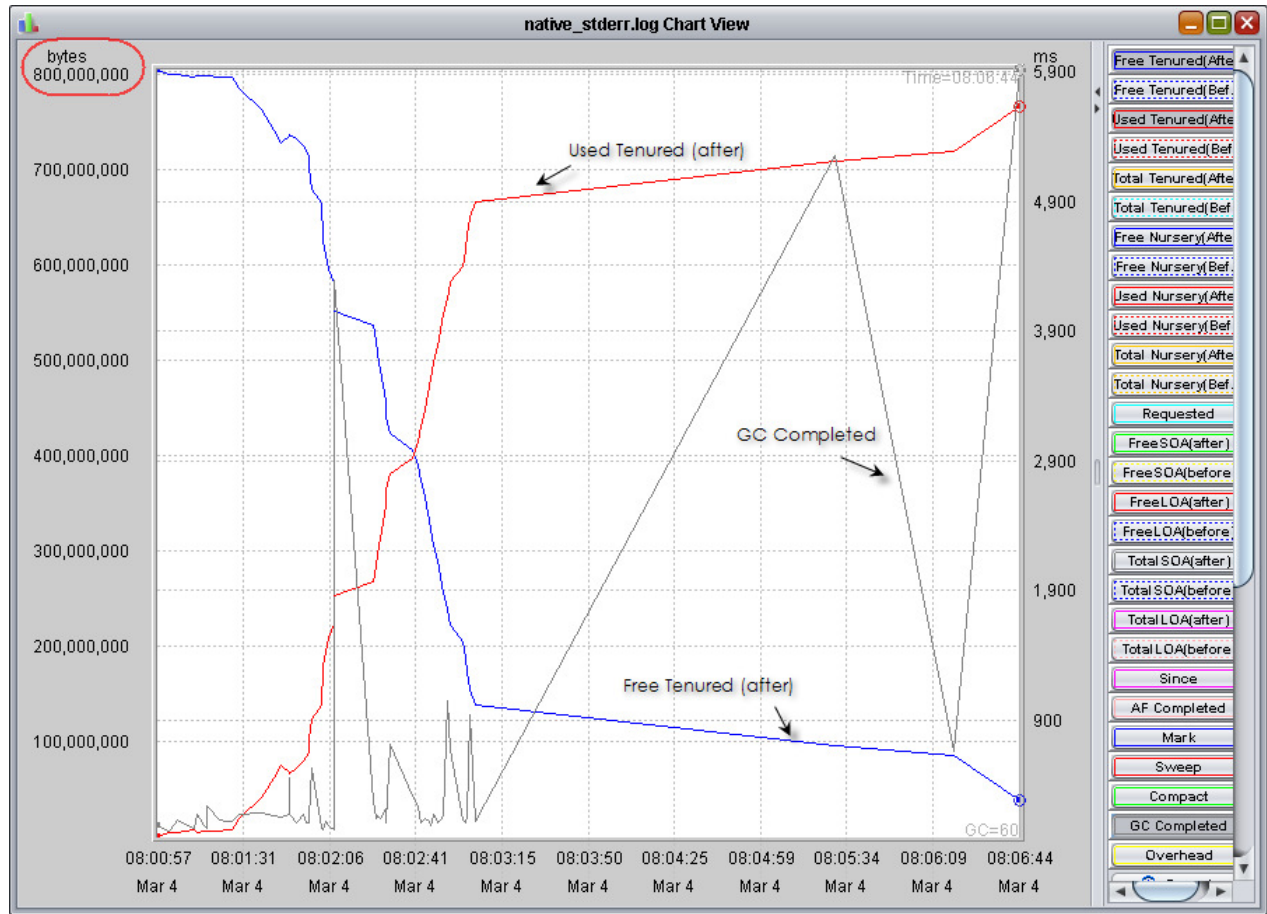
___ 2. Change the WPS maximum heap in WPS offline Mode

- ___ a. Do this in the file **server.xml** under the following directory
C:\WID7_WTE\runtimes\bi_v7\profiles\qwpw\config\cells\qcell\nodes\qnode\servers\server1
Open it via Notepad, scroll down to the **processDefinitions** section in the XML for the number 384 and change it to 1024.)

```
stderrFilename="${SERVER_LOG_ROOT}/native_stderr.log" />
pingTimeout="300" autoRestart="true" nodeRestartState="STOPPED" />
tion="true" verboseModeJNI="false" maximumHeapSize="1024" runHProf="false" debugMode="false"
essages" value="INFO" required="false" />
```

___ b. In WID, start WebSphere Process Server

- ___ c. You will notice in Wintail that WPS now recovers the last failed activity of the **SenderModule** based on exactly the same parameters of your last run. Let it run through the sending process and wait until it is finished (which might a couple minutes)
- ___ d. open Windows Explorer via the Windows quick launch bar, navigate to C:\WID7_WTE\runtimes\bi_v7\profiles\qwps\logs\server1
- ___ e. copy the file native_stderr.log to **test3.log**
- ___ 3. Use the PMAT Tool to create a graph to analyze this test result
- ___ a. open the file C:\MemoryTests\test3.log



In this graph you notice an allocation behavior without too much Garbage collection overhead to a maximum memory allocation of around 770MB of the Heap. When the asynchronous Messages are delivered the Garbage collector is able to free up all the allocations, which takes relatively long (1.4ms). But most important is that no excessive garbage collection takes place during the sending process. This increases the WPS performance for this use case tremendously. As the heap size has been tuned for this use case also no OutOfMemory Exception is thrown.

End of exercise

Exercise review and wrap-up

In this exercise you have used the verbose:gc Log to analyze the JVMs Garbage collector behavior. You have provoked an OutOfMemory exception by running a test Case which was not fitting to the JVM Memory configuration. You have changed the JVM Heap in Offline Mode, repeated the failed Test with enough memory and finally inspected the influence of a well configured Heap on Performance.

This Lab is considered to be an entry in the WPS Performance Tuning topic. If you are interested to learn more, please sign up for the official IBM Education course (**WB724** - WebSphere Process Server V7.0 Performance and Tuning)

Test Case Log – Page 1

Test Case Name: *Get used to the PerformanceTester application*

	1 KB	100 KB	1 MB
SCA sync			-
SCA async			-
JMS async			-
HTTP sync			-
WS sync			-

Test Case Name: *Configure the Resource Adapters to use new Thread Pools*

	1 KB	100 KB	1 MB
SCA sync			-
SCA async			-
JMS async			-
HTTP sync			-
WS sync			-

Test Case Name: *Tune the target module for asynchronous SCA*

	1 KB	100 KB	1 MB
SCA sync			-
SCA async			-
JMS async			-
HTTP sync			-
WS sync			-

Test Case Log – Page 2

Test Case Name: Tune the target modules for JMS bindings

	1 KB	100 KB	1 MB
SCA sync			
SCA async			
JMS async			
HTTP sync			
WS sync			

Test Case Name: Tune the server for HTTP and Web Service Bindings

	1 KB	100 KB	1 MB
SCA sync			
SCA async			
JMS async			
HTTP sync			
WS sync			